Fine-Tuning Pretrained Transformers Towards a Playstyle

Arthur Utecht
University of California, San Diego
autecht@ucsd.edu

Warren Li
University of California, San Diego
wyl003@ucsd.edu

Matthew Dalquist
University of California, San Diego
mdalquist@ucsd.edu

Haoyan Wan
University of California, San Diego
h2wan@ucsd.edu

Abstract

Chess engines, which evaluate positions and suggest moves, have become an essential part of understanding and learning chess at all levels. However, most current models, which rely on self-play and tree searches, often make "computer" moves which are difficult for humans to understand and learn from. We fine-tune a transformer pre-trained on human games using moves from he highest rated player in the world, Magnus Carlsen. A transformer can encode deep strategic understandings of the position, and by using Carlsen's games, we can obtain a high level of play without learning from unintuitive moves. We were able to improve the win ratios of CT-EFT-20 from 0.49 to 0.57, and CT-EFT-85 from 0.93 to 1.07 through fine tuning with Carlsen's games.

1 Introduction

In 1997, Deep Blue became the first chess engine to defeat a world chess champion. Since then, chess engines have become far stronger than any human. As a result, they have become an essential part of chess. They have become a core feature of every mainstream website to play chess. They allow players to play against strong engines and identify puzzles to help players learn. Perhaps most significantly, they help all players understand their mistakes and positions, such that even the strongest players review their games with the chess engine. Given the recent surge in the popularity of chess, this is more relevant now than ever.

Deep learning has become an important part of these chess engines. Typically, they integrate a deep learning model into a broader policy which uses searching. Though almost impossible to beat, these models often play inhuman moves, making them difficult to learn from. However, another method is to train a transformer to predict the next move using supervised learning. By learning exclusively from human moves, these engines can play in a more understandable way.

We propose fine-tuning a pretrained transformer using moves by Magnus Carlsen, the highest ranked chess player in the world. We will use a publicly available enlarged version of the first transformer proposed by Vaswani et. al. [6]. The model was pretrained on the Lichess Elite dataset, a collection of games by master-level players. We will fine-tune the model using pgn files of games by Magnus Carlsen.

Submitted to 37th Conference on Neural Information Processing Systems (NeurIPS 2023). Do not distribute.

2 Related Work

Early chess engines did not use deep learning. Deep Blue relies on several technological innovations, including single chip search engines, parallel processing, search extensions, complex evaluation of positions, and a Grandmaster database for early positions [1]. However, Deep Blue is not "deep" in the sense of using deep learning. Instead, it performs a complex search of a tree of possible positions with an alpha-beta search algorithm, using heuristics to determine which nodes to expand, how to assign credit, and inform two evaluation functions, one slower and more thorough than the other, to evaluate positions.

[5] Today, chess engines rely on self play with deep learning to evaluate positions. Typically, they rely on domain-specific knowledge and examples of expert play during training. On the other hand, the approach used in AlphaGo Zero has been generalized to several games, including chess [5]. This generalized algorithm uses Monte-Carlo tree search, where nodes are expanded based on move probabilities supplied by a deep network. With this approach, AlphaZero has achieved superhuman play in chess and other games.

Since such engines can achieve superhuman play using moves that would be unnatural for humans to make or calculate, they can be difficult to learn from. The application of deep learning which cannot explain its decisions exacerbates this problem. The most popular chess engine, Stockfish, has been modified to enhance usefulness to humans. ShashChess is selective in the moves it analyzes and how it analyzes those moves, relying on a formal theory of chess strategy [3]. It also uses reinforcement learning in its search function and considers the nature of the position and the strength and preferences of the human player.

In sharp contrast to search-based reinforcement learning algorithms, models have been proposed to learn policies using supervised learning. By using a transformer with a causal mask to predict the next action in a sequence, these models have achieved state-of-the-art performance in several model-free tasks [4]. Applied to chess, this method has been used to outperform AlphaZero with much less computation [2]. Importantly for helping human players, the transformer plays in a more human-like manner, able to recognize deep strategic ideas in positions.

3 Methods

We based our methods on and used code from the chess-transformers repository.

3.1 Dataset

To train our model, we used a dataset of a pgn file of 7463 games played by Magnus Carlsen provided by YottaBase. A pgn file is a file describing metadata and the moves played in a chess game. We used pgn-extract to convert the pgn file to a FENs, which describe the position on the board, for each position in each game. In a separate file, we stored the corresponding moves made in each position.

Finally, we encoded the positions where it was Carlsen's turn, who's turn it was, castling rights for each color, the next sequence moves made, and the length of the move sequence in an h5 file.

3.1.1 Data Augmentation

Due to the small size of our Carlsen dataset relative to the dataset used for pretraining, we decided to increase its size with data augmentation. Our goal of the augmentation was to give the model a different view of Carlsen's moves. We did this by flipping the board, changing the color of the team and turns, and switching the team Carlsen is playing for. In other words, if he played as white, and his pieces started at the bottom of the board, we changed him to be playing as black with his pieces starting at the top of the board. In order to keep his moves as they were in the original game, we also switched the turns so black plays first. Ideally, the augmentation allows each tower of the transformer to see a different part of the game and learn an extended set of moves from the original Carlsen set.

To generate the new data, we read each line from the FEN and Moves files, performed the augmentations, and wrote the augmented line to new files. The augmentation steps are as follows:

Flip the board in FEN:

Original: r1bqkbnr/pppp1ppp/2n5/4p3/2B1P3/8/PPPP1PPP/RNBQK1NR w KQkq - 2 3

Flipped: R1BQKBNR/PPPP1PPP/2N5/4P3/2b1p3/8/pppp1ppp/rnbqk1nr w KQkq - 2 3

Change the color of pieces (capital letters: white; lowercase: black):

Flipped: R1BQKBNR/PPPP1PPP/2N5/4P3/2b1p3/8/pppp1ppp/rnbqk1nr w KQkq - 2 3

Colors Swapped: rnbqk1nr/pppp1ppp/8/2b1p3/4P3/2N5/PPPP1PPP/R1BQKBNR b KQkq - 2 3

Change the turn:

Flipped and Swapped: rnbqk1nr/pppp1ppp/8/2b1p3/4P3/2N5/PPPP1PPP/R1BQKBNR b KQkq - 2 3

Turn Changed: rnbqk1nr/pppp1ppp/8/2b1p3/4P3/2N5/PPPP1PPP/R1BQKBNR w KQkq - 2 3

Invert moves in Moves file:

Original: e2e4

Inverted: e7e5

The result of these changes are used to create a .h5 file with both the original and augmented data.

3.2 Model Architecture

3.2.1 CT-EFT-20

Our first model was a pretrained version of the CT-EFT-20 encoder from the chess-transformers repository, which predict the "From" and "To" squares of the best next half-move given a position. This is the same encoder used in [6]. This model uses an embedding size of 512 and has 8 attention heads. Query, key, and value vector sizes are 64. Hidden layers in the inner feedforward have dimension 2048, and there are 6 layers. It has close to 20,000,000 parameters.

3.2.2 CT-EFT-85

We used the pretrained version of the CT-EFT-85 encoder from the chess-transformers repository, which predict the "From" and "To" squares of the best next half-move given a position. This model uses an embedding size of 768 and has 12 attention heads. Query, key, and value vector sizes are 64. Hidden layers in the inner feedforward have dimension 3072, and there are 12 layers. It has close to 85,000,000 parameters.

3.3 Training

We trained both models on an NVIDIA A6000. For both models, we initialized the model using the existing pretrained weights. Then we trained the CT-EFT-20 and CT-EFT-85 models on our dataset of Magnus Carlsen moves to try and improve its performance.

3.3.1 CT-EFT-20

We used batch sizes of 512 for CT-EFT-20. We used Cross Entropy Loss with label smoothing of 0.1. There was a dropout of 0.1. We updated parameters every 4 steps and used 8000 warmup steps. We used an Adam optimizer with $\epsilon = 10^{-9}$ and β's 0.9 and 0.98, and we kept the learning rate schedule from [6] (beginning with a learning rate of 0.00027950849718747374). We trained this model for 164 epochs.

3.3.2 CT-EFT-85

We used batch sizes of 512 for CT-EFT-85. We used Cross Entropy Loss with label smoothing of 0.1. There was a dropout of 0.1. We updated parameters every 4 steps and used 8000 warmup steps. We used an Adam optimizer with $\epsilon = 10^{-9}$, β's 0.9 and 0.98, and an exponential decay learning

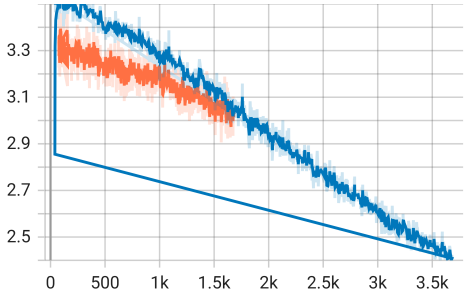
rate schedule (starting with a learning rate of 0.0625 and using a decay rate of 0.06). We trained this model for 54 epochs.

3.3.3 CT-EFT-20 With Data Augmentation

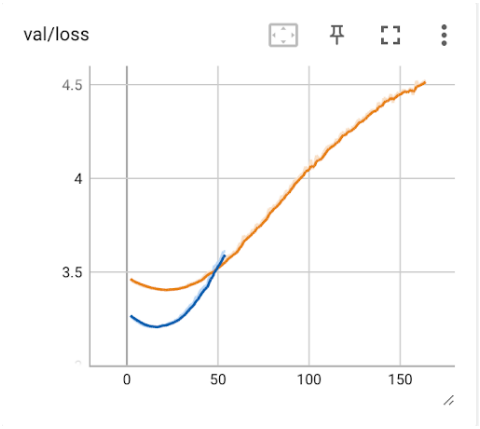
To train the CT-EFT-20 with data augmentation, we used the same hyperparameters as the original CT-EFT-20 - we just used the augmented dataset. Due to time constraints, we trained this model for 37 epochs, starting from the best performing epoch 28 checkpoint that we had from training on the original dataset.

4 Results

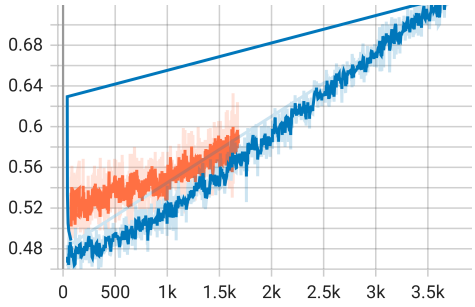
4.1 Training Performance



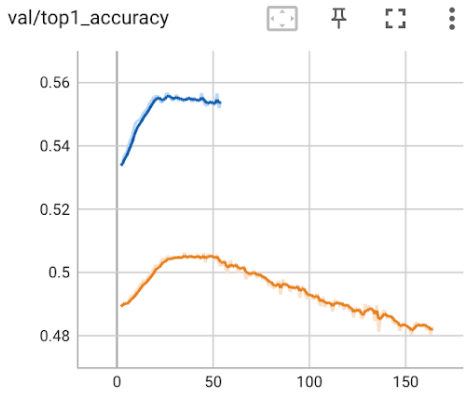
(a) Training loss per step for CT-EFT-20 (blue) and CT-EFT-80 (orange)



(b) Validation loss per step for CT-EFT-20 (blue) and CT-EFT-80 (orange)



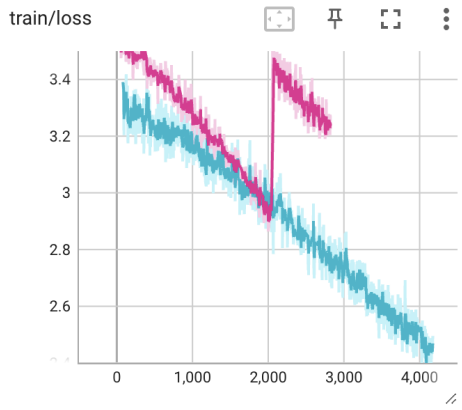
(c) Training top one accuracy vs. step for CT-EFT-20 (blue) and CT-EFT-80 (orange)



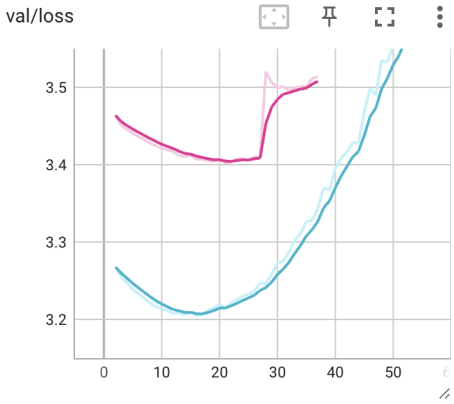
(d) Validation top one accuracy vs. step for CT-EFT-20 (blue) and CT-EFT-80 (orange)

Figure 1: Training Performance metrics.

After training both the CT-EFT-20 and CT-EFT-85 models for 164 and 54 epochs respectively, we can see that the training loss decreases at a steady rate while the training accuracy increases. However, from the validation loss, we see that both models start having an increase in validation loss around epoch 25-30. This suggests that the models have started overfitting the training set. This is likely due to the model complexity and smaller dataset size. Thus, for the evaluation of competitive performance for our models, we used the checkpoints at epoch 28 for CT-EFT-20 and epoch 25 for CT-EFT-85, which had the best accuracy on the validation set.



(a) Training loss vs. step for CT-EFT-20 with data augmentation (pink)



(b) Validation loss vs. step for CT-EFT-20 with data augmentation (pink)

Figure 2: Data augmentation performance for CT-EFT-20.

To try and mitigate the overfitting on the dataset, we tried data augmentation. The CT-EFT-20 model had a jump in training and validation loss at the epoch 28 mark, which is when we started training it on the augmented dataset. From the graphs, we can see that the training loss steadily decreases on the augmented dataset, but the validation loss increased and did not go down. This suggests that the data augmentation was not effective to combat the overfitting on the dataset. In the future, it might be worth trying more techniques involving greater dropout and regularization in the model itself, or collecting a bigger dataset.

4.2 Competitive Performance

Elo Rating System (Skill Level) **Mathematical Formulation:**

- Final ELO: $R' = R + K(S - E)$
  - R': New rating, R: Old rating
  - K: Scaling constant (usually 30)
  - S: Actual score (1 = win, 0.5 = draw, 0 = loss)
  - E: Expected score: $E = \frac{1}{1+10^{\frac{R_{\text{opponent}} - R}{400}}}$
- Encourages beating stronger opponents and penalizes losses to weaker ones.
- Normalized and widely adopted in competitive games for benchmarking.

Win-Loss Ratio and Win Percentage

- Win-Loss Ratio: $\frac{\text{Wins}}{\text{Losses}}$
- Win Percentage: $\frac{\text{Wins}}{\text{Total Games}} \times 100$
- Raw indicators of performance but do not consider opponent difficulty or move quality.

We evaluated both models against Stockfish (Strength 6):

Elo Estimates

- Small Model (tuned): **1827**
- Large Model (tuned): **1936**

Model	Wins	Losses	Draws	Win-Loss Ratio	Win Percentage
Large Model (pretrained)	41	44	15	0.93	41%
Small Model (pretrained)	27	55	18	0.49	27%
Large Model (finetuned)	45	42	13	1.07	52%
Small Model (finetuned)	30	53	19	0.57	36%

Table 1: Model performance results

4.3 Key Findings

- **Larger Models Perform Better:** The CT-EFT-85 consistently outperforms CT-E-20 across all metrics. This aligns with findings in NLP and vision domains.
- **MCTS Parameters Matter:** Properly tuning simulation count, exploration constant, and search depth significantly impacts move quality.
- **Strong Results with Limited Resources:** Even partial fine-tuning showed substantial gains. Pretraining on expert data helped generalization.
- **Transformer Models Are Viable for Chess:** Our model exhibited human-like strategy and competitive gameplay despite not relying on brute-force search.

4.4 Metric Interpretation

Large Model

- **ELO Rating:** 1936 — comparable to an experienced club player.
- **Win-Loss Ratio:** 1.07 — near parity with Stockfish 6.
- **Win Percentage:** 52% — slightly better than even performance.

Small Model

- **ELO Rating:** 1827 — clearly weaker, but still respectable.
- **Win-Loss Ratio:** 0.57 — often defeated by Stockfish.
- **Win Percentage:** 36% — consistent with the above.

Takeaways

- The larger model demonstrates superior play on all evaluation metrics.
- It competes effectively with a strong chess engine and serves as a viable tool for training or stylistic gameplay.
- Future analysis could benefit from more granular move-quality evaluation metrics.

5 Discussion

Our finetuning of the chess transformers demonstrated superior performance to their untuned counterparts. Both models saw an improvement in win ratio and Elo rating when tested against Stockfish at strength 6. Exposing the model to moves from the world's top chess grandmaster improved its ability to win chess games.

During training, we observed both models overfitting during finetuning. Loss decreased slightly before rising again. Similarly, top-1, top-3, and top-5 accuracy increased only slightly before decreasing.

This small increase did translate to improved performance - capturing the state of the model at its lowest loss allowed us to achieve better performance in gameplay, but perhaps not as strong as it could be. Specifically, the fine-tuned CT-EFT-20 won 42.3% more games than its pretrained counterpart against Stockfish, and CT-EFT-85 won 26.8% more games than its pretrained counterpart. The

significance of a small improvement in accuracy is not surprising - at a high level, a single mistake can determine the course of the entire game.

The CT-EFT-20 performance improving more noticeably may be due to the small dataset. With fewer parameters, the model may have been forced to learn the principles behind the moves without being able to "memorize" the moves themselves.

We hypothesized that the small size of the Carlsen dataset was a reason for overfitting, so we attempted data augmentation by flipping the board. This did not fix the overfitting problem - even with the augmentation, the dataset was much smaller than other datasets of chess games. Interestingly, the model performance was significantly worse in earlier epochs. This may be because the model the pretrained model had not generalized to flipped positions, or because some flipped positions (e.g. black to move in the starting position) are impossible or highly unlikely. Training for more epochs would likely mitigate this.

Adding data from other games into the training set is one way to benefit from the small dataset of Carlsen's moves without losing generalization to other positions. By training with the Carlsen dataset while at the same time introducing games from other datasets, we could force the model maintain an understanding of both the principles behind Carlsen's play and general chess play in more varied positions. For example, after every game from the Carlsen dataset, we could input and backpropogate from a position from another dataset such as the much larger LE22ct dataset.

One consideration from our finetuning methods is that it is impossible for our model to achieve superhuman performance by introducing Magnus Carlsen's moves. A theoretical 0 loss in training would mean that the model learns to play just like Carlsen, but not surpass him. It would take the move he would make in every situation instead of learning to beat him. Further reinforcement learning would be necessary for the chess transformer to surpass human performance.

For future work, it might be worth investigating techniques to mitigate overfitting on the dataset. Additionally, using this as a baseline model to further increase performance through Reinforcement Learning could have potential.

6 Authors' Contributions

Arthur: Wrote Introduction and Related Work, and part of Methods. Helped with other sections. Wrote script to isolate Carlsen's moves and modified code to use these moves as targets.

Warren: Trained both the CT-EFT-20 and CT-EFT-85 models to achieve increased performance. Contributed to Methods training and results sections of the paper.

Matthew: Started the presentation and put together roughly half of the material. Coded data augmentation steps and generated .h5 file. Contributed to Methods and Discussion sections.

Haoyan:
Wrote, implemented and ran the evaluation metrics, tests and calculated ELO. Review and proof read most of report and presentation. Provided debugging steps.

References

[1] Murray Campbell, A.Joseph Hoane, and Feng hsiung Hsu. Deep blue. *Artificial Intelligence*, 134(1):57–83, 2002.

[2] Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Michael Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. Decision transformer: Reinforcement learning via sequence modeling, 2021.

[3] Andrea Manzo and Paolo Ciancarini. Enhancing stockfish: A chess engine tailored for training human players. In Paolo Ciancarini, Angelo Di Iorio, Helmut Hlavacs, and Francesco Poggi, editors, *Entertainment Computing – ICEC 2023*, pages 275–289, Singapore, 2023. Springer Nature Singapore.

[4] Daniel Monroe and Philip A. Chalmers. Mastering chess with a transformer model, 2024.

[5] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. Mastering chess and shogi by self-play with a general reinforcement learning algorithm, 2017.

[6] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023.